VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Stereo korespondence

Stereomatch

2010 **Petr Bystrzycki**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Bystrzycki**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Stereo korespondence
Stereomatch

Zásady pro vypracování:

Stereo korespondence je jedním ze základních problémů v oblasti rekonstrukce 3D scény. Jedná se o těžký problém, který doposud není uspokojivě vyřešen. Nalezení korespondence se často formuluje jako globální optimalizační problém, který zahrnuje všechna možná geometrická a fotometrická omezení jako je například zákryt nebo průhled.

Seznamte se s algoritmy stereo korespondence. Implementujte vybraný algoritmus pro stereo korespondenci.
Navrhněte metodu porovnávající jejich úspěšnost pro zadané dvojice snímků.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.
http://vision.middlebury.edu/stereo/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Krumnikl**

Datum zadání: 20.11.2009
Datum odevzdání: 07.05.2010

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010

Petr Bystrzycki

Tímto bych rád poděkoval hlavně svému vedoucímu panu Ing. Michalu Krumniklovi za jeho rady, připomínky a čas, který věnoval při tvorbě mé práce. Dále bych chtěl poděkovat všem blízkým za velkou podporu.

Abstrakt

Předkládaná bakalářská práce se zabývá problémem stereo korespondence. Jedná se o hledání všech shodných párů pixelů v obrazech, zobrazujících jednu scénu z více úhlů pohledů. V současné době existuje mnoho metod, které řeší problém stereo korespondence. Cílem této práce je seznámení s těmito metodami a následná implementace jedné z nich. Implementována metoda je založená na dynamickém programování. U této metody byl podrobně popsán princip a postup řešení. Následně bylo provedené testování, které ověřilo rychlost a kvalitu této metody.

Klíčová slova

Stereo korespondence, Disparitní mapa, Dynamické programování, Disparita

Abstract

This bachelor work deals with the stereo correspondence problem. This problem can be described as finding all pairs of identical pixels in images, which shows a scene from multiple angles. Currently, there are many methods to solve the stereo correspondence problem. The aim of this work is to introduce these methods and choose one of them to implement. Implemented method is based on dynamic programming. This method was described in detail to explain the principles and solution. It is followed by testing to verify the quality and speed of this method.

Keywords

Stereomatch, Disparity map, Dynamic programing, Disparity

Seznam použitých zkratek a symbolů

CVPR	Conference on Computer Vision and Pattern Recognition
ICCV	International Conference on Computer Vision
IJCV	International Journal on Computer Vision
SAD	Sum of Absolute Differences
SSD	Sum of Squared Differences

Obsah

1 Úvod

Člověk může vnímat svět prostřednictvím zraku, sluchu, hmatu, chuti a čichu. Zrak je smysl, který nám umožňuje vnímat světlo, různé barvy a tvary. Pro člověka je to smysl nejdůležitější, protože 80% všech informací získáváme právě tímto smyslem. Zrak je zaměřen především na vnímání kontrastu, proto dovoluje vidění obrysů předmětů, jejich vzdálenost a významně se podílí na orientaci v prostoru [11]. Lidské oko je párový orgán, který nám umožňuje vidět. Když si zkusíme zakrýt postupně obě oči, zjistíme, že každé oko vidí obraz z jiného úhlu a tím pádem nám do mozku přicházejí dva odlišné obrazy, které mozek musí zpracovat. Tuto schopnost skládání obrazů dohromady se mozek naučí, když jsme malé děti. Jinak bychom viděli dvojitě. Díky této schopnosti dokážeme vnímat předměty prostorově.

Protože počítače a výpočetní technika nechtějí zaostávat, snaží se člověka napodobit ve všech oborech, včetně zraku. Tato vlastnost je důležitá pro oblast robotiky, kde se snaží vytvořit roboty, kteří se budou bezpečně pohybovat v neznámém prostoru. Místo očí používají kamery a mozek nahradíme výkonným procesorem. Teď už nám chybí jen schopnost zpracování vstupních obrazů, aby mohli roboti také vnímat prostorově. Musíme tedy navrhnout metodu, která nám ze dvou 2D obrazů vytvoří 3D obraz. Při návrhu metody je kladen důraz nejen na kvalitní zpracování obrazů, ale také na jeho rychlosti. Tato technologie se používá nejen v oblasti robotiky, ale také pro zobrazení povrchů planet ze satelitů. Další použití nachází v dopravě (sledování aut, automatické řízení aut atd.).

Navržení správné metody a následné zpracování obrazů doprovází řada problémů. My si tyto problémy obecně a chronologicky popíšeme. Protože existuje mnoho algoritmů, které řeší problém stereo korespondence, je potřeba tyto metody rozdělit a některé si více popsat. Poté se budeme věnovat jedné metodě, kterou jsme si zkusili naimplementovat. Jedná se o metodu, založenou na dynamickém programování. Podíváme se na princip této metody a popíšeme si, jakým způsobem jsme metodu naimplementovali. Poté metodu otestujeme na vstupních obrazech a zhodnotíme výsledky.

Správnost výsledků metod stereo korespondence ovlivňuje také kvalita vstupních obrazů, které tyto metody zpracovávají. Proto se podíváme, jakým způsobem můžeme obrazy dobře pořídit a nebo jakým způsobem je správně upravit. Avšak některé chyby nám mohou vzniknout už při samotném uložení obrazů z reálného světa do počítače. My si tento proces vysvětlíme v následující kapitole nazvané „Snímání“.

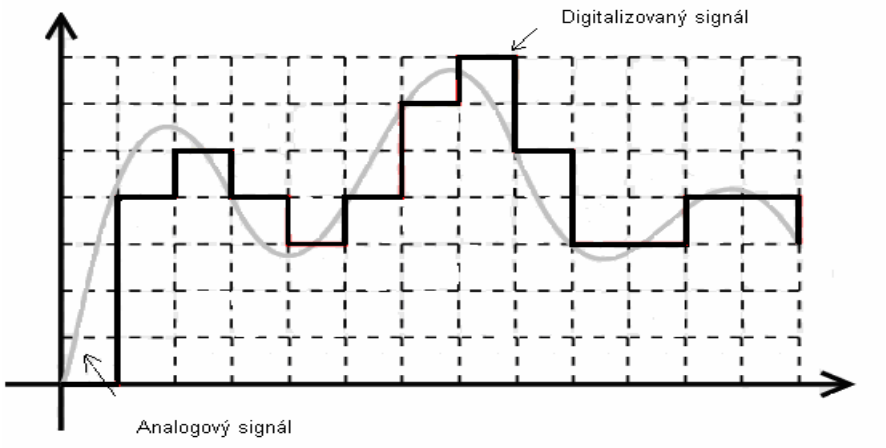
2 Snímání

Všechny události, které se odehrávají v reálném světě, můžeme zachytávat našimi smysly. Tyto události se neustále mění, jedná se o nekonečně proměnlivá data. Lidé tato data přijímají po celý život. V našem případě jsou data myšlena jako zvuky, barvy, vůně, tvary atd..

Na druhé straně máme počítače, které nejsou schopny vnímat reálný svět jako my. Jsou schopny příjímat data pouze v binárním kódu. To znamená, že rozumí pouze 1 (zapnuto) a 0 (vypnuto). Přesto jsme dosáhli v tomto oboru neskutečných pokroků. Počítače jsou schopny přehrát video, hudbu a zobrazit obrázky. Aby byly schopny toto provést, musíme tedy převést data z reálného světa (analogový signál) na jejich jazyk (signál digitalizovaný), tedy do binárního kódu. Tomuto kroku říkáme digitalizace. Skládá se ze tří kroků: vzorkování, kvantizace a kódování. My si popíšeme pouze první dva, které nám ovlivňují výsledky při zpracování obrazu metodou stereo korespondence.

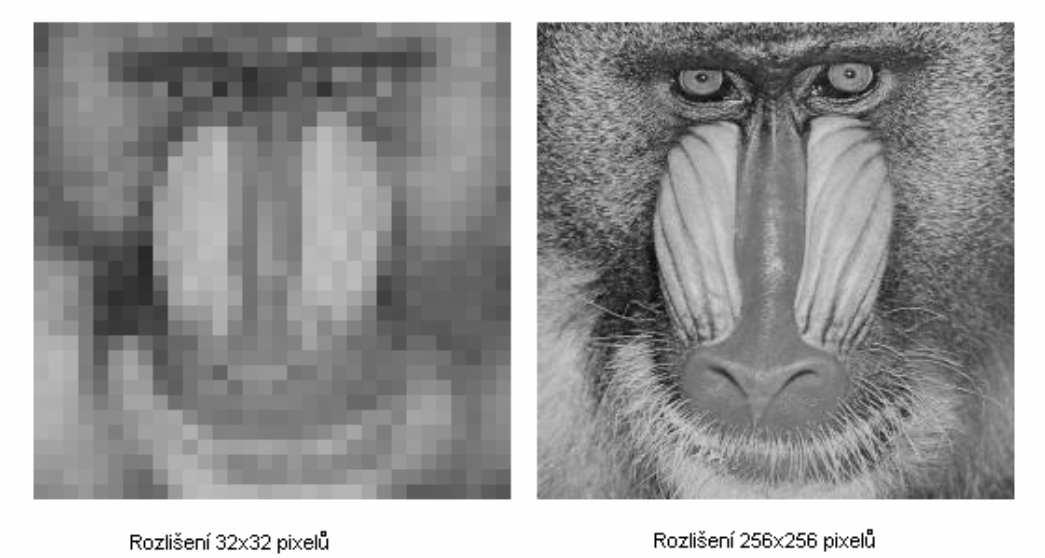
2.1 Vzorkování

Vzorkováním rozumíme, že budeme odebírat hodnoty (vzorky) ve stejných intervalech. Původní analogový signál je přetvořen na posloupnost okamžitých hodnot. Digitální obraz získáme pomocí vzorkování obrazu do matice MxN a kvantováním do K úrovní [5]. Na obrázku 2.1 [16] vidíme výsledný digitalizovaný signál (po vzorkování i kvantizaci).



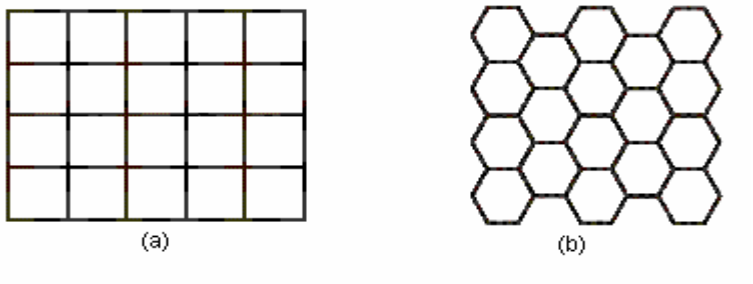
Obrázek 2.1 Rozdíl mezi signály

Volba rozlišení je jeden z nejdůležitějších kroků digitalizace. Při nízkém rozlišení (malá velikost matice MxN) ztrácíme detaily v obraze. Když zvolíme moc velké rozlišení, budeme mít sice detailní obraz, ale výrazně nám stoupne náročnost výpočtů při zpracování obrazu metodami stereo korespondence. Ukážeme si vliv rozlišení na detaily v obraze viz obrázek 2.2 [6]. Rozlišení obrázků se uvádí v DPI (počet bodů na palec).



Obrázek 2.2 Rozdíl v rozlišení

Jednomu vzorkovacímu bodu odpovídá v obraze po digitalizaci jeden obrazový bod (pixel). Tyto pixely se při vzorkování řadí do vzorkovací mřížky. Nejčastěji používané mřížky jsou čtvercová a hexagonální. Jejich strukturu můžeme vidět na obrázku 2.3. Každá z nich má své výhody a nevýhody. V našich vstupních obrazech byla použita čtvercová mřížka. Vychází z konstrukce většiny snímacích prvků a je velmi snadno realizovatelná. Avšak oproti hexagonální mřížče má své nevýhody, týkající se měření vzdálenosti a spojitosti objektů.

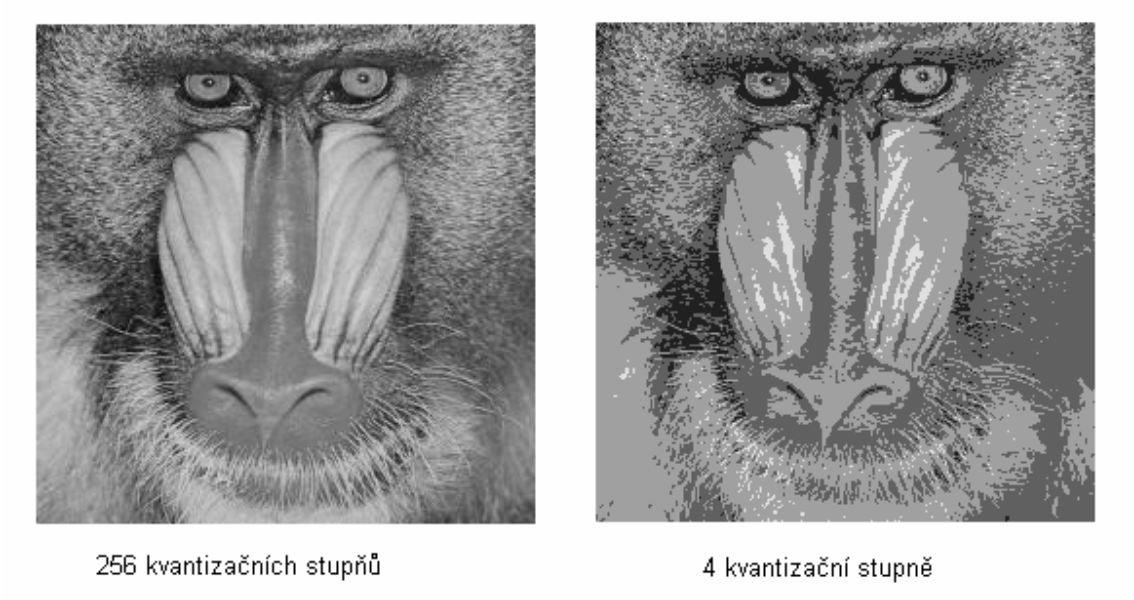


Obrázek 2.3 Vzorkovací mřížky

2.2 Kvantizace

Kvantizace je dalším krokem digitalizace. Naměřené hodnoty jasu jsou reálná čísla. My však pro práci s obrazy používáme celočíselný rozsah jasu 0-255 (reprezentujeme 256 různých odstínů šedi). Proto musíme vzorky rozdělit podle velikosti do kvantizačních stupňů tak, že naměřené vzorky prostě zaokrouhlíme. Uvedeme si příklad. Máme dva vzorky, které mají hodnotu 2,1 a 2,4. Při převodu do kvantizačních stupňů se nám tyto dva vzorky uloží do stejné kvantizační úrovně. Přiřadí se jim tedy hodnota 2. Tento jev nazýváme kvantizační zkreslení [8]. Aby nám toto zkreslení neovlivnilo další zpracování digitalizovaného signálu, je třeba

vhodně zvolit kvantovací úrovně. Podívejme se na vliv kvantizace na výsledný obraz viz obrázek 2.4 [6].



Obrázek 2.4 Vliv kvantizace

Nyní víme, jakým způsobem se nám ukládají obrazy do počítače a jak hodně ovlivňují pořízené obrazy. Uveďme další problémy, které vznikají při vytvoření obrazů a úpravy, které musíme provést, než použijeme naši metodu stereo korespondence. Problémy si dále rozdělíme na dvě fáze:

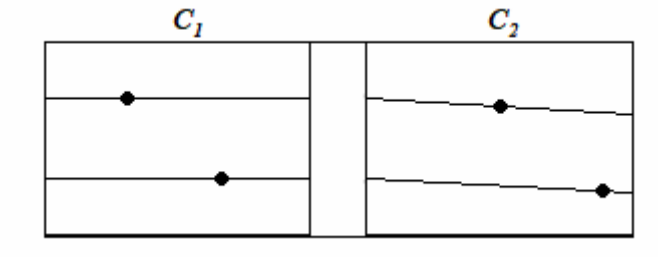
- Problémy vznikající při špatném nastavení kamer
- Problémy při spárování odpovídajících pixelů a následné vytvoření disparitní mapy

2.3 Pořízení obrazů

Obrazy získáme buď jednou pohybující se kamerou, nebo musíme mít statickou strukturu minimálně ze dvou kamer. Už kalibrace kamer, které nám poskytnou obrazy, je důležitou součástí pro jakékoli další zpracování obrazu. Některé metody stereo korespondence jsou schopny pracovat s informacemi pořízenými z nekalibrovaných kamer, avšak kalibrace kamer je důležitým krokem ve všech oblastech vyžadujících metrickou informaci [9]. Tímto bych chtěl poukázat na to, jak těžký problém stereo korespondence je, protože můžete mít naimplementovanou výbornou metodu stereo korespondence, avšak když získáte špatné vstupní obrazy, stejně výsledek bude nepřesný.

Podívejme se nejprve na situaci viz obrázek 2.5. Máme k dispozici obrazy C_1 a C_2 a potřebujeme najít k pixelům, které jsou vyznačeny v obraze C_1 odpovídající pixely, které se

nacházejí v obraze C_2. Protože druhý obraz není pořízen ve správném úhlu, řádek ve kterém se nachází pixel v obraze C_1, neodpovídá řádku v obraze C_2. Toto nám velmi zpomalí vyhledávání těchto pixelů pomocí metod stereo korespondence, protože hledaný odpovídající pixel se může nacházet kdekoli v obraze C_2, takže musíme hledat odpovídající párový pixel po celém obraze.

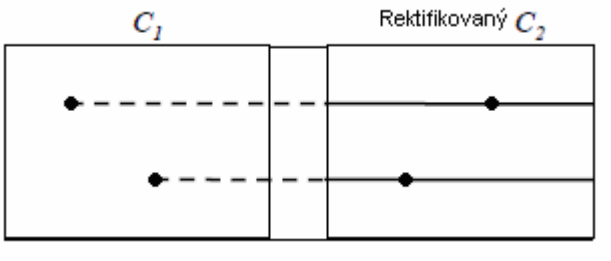


Obrázek 2.5 Obrazy před rektifikaci

Většina metod stereo korespondence vyžaduje pro svůj správný chod, aby pixel, pro který hledáme příslušný pixel v obraze druhém, se nacházel ve stejném řádku. Takhle dobře připravených obrazů dosáhneme dvěma způsoby:

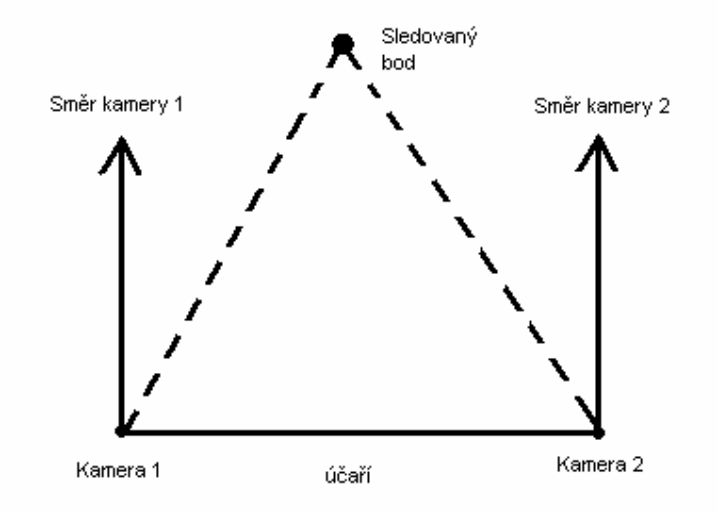
- Pomocí epipolární geometrie
- Pomocí speciálního nastavení kamer

Díky závěrům epipolární geometrie můžeme vstupní obrazy rektifikovat pro následnou metodu stereo korespondence. Jedná se o transformaci vstupních obrazů tak, aby odpovídající pixely ležely ve stejném řádku v obou obrazech (viz obrázek 2.6). Dosažení této transformace výrazně urychlí chod použité metody stereo korespondence, protože nyní nemusíme hledat odpovídající pixel v celém obraze, ale pouze v daném řádku. Vyhledávání je pouze 1-D místo 2-D.



Obrázek 2.6 Obrazy po rektifikaci

Nyní se podíváme na druhý způsob, kdy nemusíme vstupní obrazy transformovat, abychom dosáhli stejného výsledku. Jedná se o speciální paralelní nastavení kamer, kde jsou roviny sítnic nastaveny horizontálně a obě kamery mají identickou ohniskovou vzdálenost. Jak vypadá takové nastavení, můžeme vidět na obrázku 2.7.



Obrázek 2.7 Paralelní nastavení kamer

Na tomto paralelním nastavení kamer je důležitá velikost účaří mezi dvěma kamerami. Tato vzdálenost nám ovlivňuje výsledek metody stereo korespondence. Při větší vzdálenosti účaří získáme lepší zobrazení hloubky, avšak použití následné metody stereo korespondence bude náročnější, protože mnoho objektů se bude v obraze překrývat (tyto objekty nazýváme occlusion objekty). Occlusion objekty způsobují největší počet chyb při zpracování obrazů vybranou metodou stereo korespondence. Při menší velikostí účaří je to přesně naopak. Obrazy budou mít málo společných objektů, výsledek metody stereo korespondence bude přesnější , ale zobrazení hloubky bude horší.

Pro získání přesnějších výsledků můžeme použít více než dva vstupní obrazy. Pokud pracujeme s více než se dvěma obrazy, musíme si jeden určit jako referenční. Potom vytvoříme disparitní mapy (viz kapitola 2.4) se všemi ostatními obrazy a tímto referenčním obrazem. Výsledné disparitní mapy pak spojíme dohromady [7]. Výsledkem bude hustá disparitní mapa. Tento způsob nám redukuje neviditelná místa, protože máme díky více kamerám širší rozsah pohledů. Další velkou výhodou při použití více obrazů je to, že více pohledů redukuje nejasnosti v párování obrazů a výsledná disparitní mapa bude lépe zobrazovat hloubku objektů. Ale musíme si uvědomit, že potřebujeme spočítat více disparitních map, proto bude tento způsob výpočetně náročnější a vznikne nám více problémů při použití metody stereo korespondence.

2.4 Volba disparitní mapy

Pokud máme připravené obrazy, je třeba rozhodnout, kterou metodu stereo korespondence zvolíme. Úkolem těchto metod je vytvořit ze vstupních obrazů disparitní mapu. Disparitní mapa je matice D o rozměrech MxN, kde M je výška vstupního obrazu v pixelech a N je šířka vstupního obrazu v pixelech. Hodnota prvku d_{xy} se rovná vzdálenosti pixelu o poloze x, y v referenčním obraze od jeho korespondujícího pixelu v obraze druhém. Pomocí disparitní mapy jsme schnopni následně vypočítat skutečnou vzdálenost nějakého obrazového bodu od

snímacího zařízení. Jsme tedy schopni určovat vzdálenost objektů, mezi objekty a podobně. Metody stereo korespondence si rozdělíme na dvě hlavní skupiny podle typu zpracování [14].

- Příznakové: Výsledkem je řídká disparitní mapa
- Korelační Výsledkem je hustá disparitní mapa

2.4.1 Příznakové zpracování obrazů

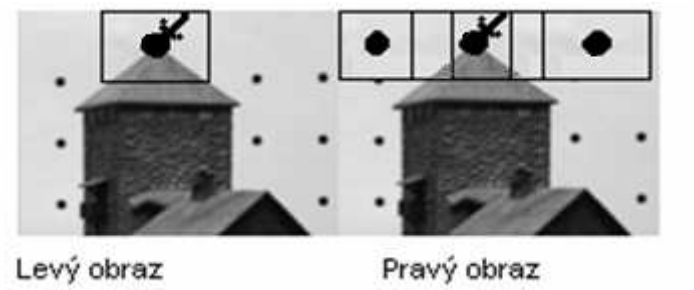
Při tomto zpracování vybíráme z jednoho obrazu pouze objekty (např: roh, hranu, strukturu..) viz obrázek 2.5, které následně hledáme v druhém obraze. Tyto objekty se dají detekovat, protože v těchto oblastech dochází ke prudké změně jasu pixelů. Následné spárování pixelů probíha velmi rychle.

Obrázek 2.8 Ukázka výběru objektů

Toto řešení je vhodné pro použití v robotice, kde potřebujeme zpracovávat obrazy v reálném čase. Dále si umí dobře poradit s obrazy, ve kterých se nachází mnoho překrytých hran. S tímto jevem má problémy korelační zpracování, které v takových místech vypočítává chybné disparity. Pokud se nám stane, že při snímání obrazů dojde ke změně osvětlení, je lépe použít právě toto zpracování, jelikož příznakové zpracování obrazu je relativně necitlivé na změny osvětlení v obraze. Také je vhodné použití tohoto zpracování pro obrazy, které obsahují silné hrany. Abychom nezůstali pouze u výhod, musíme se podívat také na jednu velkou nevýhodu. Oproti korelačnímu zpracování, tyto metody vytváří pouze řídkou disparitní mapu, která nám neposkytuje detailní zobrazení objektů. Z tohoto důvodu využíváme toto zpracování obrazů pouze tam, kde nepotřebujeme znát detailní informace o objektech, ale stačí nám pouze vědět, kde se objekty nacházejí, nebo jak daleko jsou od sebe vzdálené (orientace v prostoru). Další nevýhoda v příznakovém zpracování obrazů může nastat při hledání korespondujících objektů (Jak změříme podobnost dvou hran, nějakých rohů , nebo celých struktur ?).

2.4.2 Korelační zpracování obrazu

Oproti příznakovému zpracování obrazů, nám tyto metody párují všechny možné pixely na obrázku. Na každém pixelu si definujeme velikost okolí, ve kterém se hledaný bod nachází uprostřed a poté toto okolí hledáme v obraze druhém (viz obrázek 2.6). Tento postup si podrobněji popíšeme později.

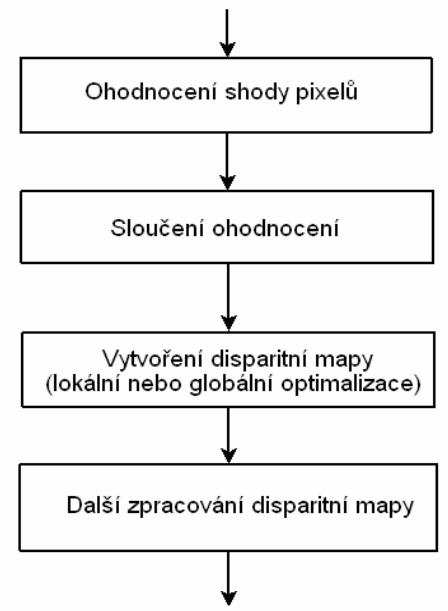


Obrázek 2.9 Korelační metoda

Velkou výhodou těchto metod je jednoduchá implementace. Výsledkem korelačního zpracování obrazů je hustá disparitní mapa. Tato mapa nám poskytne detailní informace o objektech v obrazech (ulehčí nám rozpoznání objektů v obraze). Aby toto zpracování dobře pracovalo, potřebujeme obrazy s detailními texturami. Dále bychom neměli používat korelační zpracování pokud máme k dispozici vstupní obrazy, které jsou pořízeny z velmi různých míst. U takových obrazů dochází ke změně osvětlení. Toto by nám velmi zhoršilo správnost výsledků metod stereo korespondence. Další faktor, který nám ovlivňuje výsledky, je velikost okolí hledaného pixelu. Proto musíme zvolit správnou velikost okolí. Dopady při špatném zvolení velikosti okolí jsou popsány v kapitole 3.1.1. Efektivně naimplementované metody korelačního zpracování jsou výrazně pomalejší, než metody v příznakovém zpracování.

3 Metody stereo korespondence

V předchozích letech bylo vyvinuto a naimplementováno mnoho algoritmů stereo korespondence. I když se liší v mnoha aspektech, pro mnoho z nich můžeme rozlišit obecné charakteristické fáze algoritmů. Tyto fáze máme zobrazené na obrázku 3.1 [4]. Metody si dále rozdělíme podle způsobu zpracování obrazu na **lokální a globální**.



Obrázek 3.1 Schéma fázi algoritmů

První fáze ohodnocení pixelů je relativně přímočará. V této fázi měříme podobnost pouze mezi dvěma pixely (*pixel based*), nebo měříme podobnost dvou okének (*window based*). Okénky máme na mysli nějaké čtvercové okolí kolem pixelu. Určit shodu pixelů můžeme pomocí lokálních metod, které máme popsány níže.

V druhé fázi dochází ke sloučení výsledků všech vypočtených ohodnocení. Tato fáze probíhá pouze u lokálních metod. Globální metody tuto fázi přeskakují. Způsoby sloučení a jejich vlastnosti jsou popsány v [4].

Ve třetí fázi vypočteme výslednou disparitní mapu. Tento krok může být organizovaný jako lokální optimalizace, nebo jako globální optimalizace. V lokální optimalizaci výpočet výsledných disparit probíhá jednoduše. Prostě zvolíme pro každý pixel tu disparitu, která měla nejnižší hodnotu ohodnocení (získanou v druhé fázi). Tento princip se nazývá „vítěz bere vše“ (*winner-také-all*). Proces globální optimalizace se dá popsat jako nalezení minimum funkce ohodnocení. Úkolem je tedy najít disparitní funkci d (cenová funkce), která minimalizuje globální energii pomocí následujícího vzorce (viz rovnice 3.1). Datová podmínka $E_{data}(d)$

měří jak dobře disparitní funkce *d* odpovídá se vstupními obrazy [15] (zda jsou si pixely podobné). Protože se ukázalo, že minimalizace této cenové funkce je NP-těžký problém, musely se vyvinout efektivní algoritmy (globální metody).

$$E(d) = E_{data}(d) + \lambda E_{smooth}(d)$$

Rovnice 3.1 Funkce minimalizující globání energii [15]

V poslední fázi zpracujeme výslednou disparitní mapu. Tato fáze je zaměřena na vylepšení kvality výsledné disparitní mapy. Například použití filtru, abychom se zbavili rozostřenosti objektů v obraze. Další způsob spočívá v doplnění disparit do míst, ve kterých jsme nebyli schopni určit disparitu, nebo disparitní hodnota nebyla přijmuta během fáze ověřování. Ostatní způsoby vylepšení kvality disparitní mapy jsou popsány v [4].

3.1 Lokální metody

Tyto metody používáme především v korelačním zpracování, ale mohou být použity i pro příznakové zpracování. Vypočet výsledných disparit je jednoduché. Na každém pixelu vybereme disparitu, která měla nejmenší ohodnocení (nebo největší, záleží na použité metodě např. cross-correlation). Omezení těchto metod a také dalších metod stereo korespondence spočívá v tom, že unikátní spárování pixelů je vykonáno pouze pro jeden obraz (referenční obraz), zatímco ostatní pixely na druhém obraze mohou být spárovány s více body. Pro příklad uvedu, že pokud si zvolíme referenční obrázek levý, pixely, které chceme spárovat vybíráme pouze v levém obrázku a odpovídající pixely hledáme v pravém. Vyjmenujme si pár typických zástupců těchto metod:

- SSD (Sum of Square Difference)
- Adaptive Window
- Multiple window algorithm
- SAD (Sum of absolute Difference)

3.1.1 Sum of Square Differences

Na tuto metodu se podíváme blíže, neboť ji používáme v našem naimplementovaném programu. Jelikož pixely mají podobnou hodnotu jasu, je těžké rozhodnout, která dvojice pixelů se má spárovat. S tímto problémem nám pomůže právě metoda SSD. Princip této metody spočívá v tom, že se ve zvoleném okénku vytvořeném okolo pixelu, porovnají jasy pixelů a ohodnotí se jejich podobnost (tzn. nekontroluje pouze jasy dvou pixelů, ale i jejich okolí pro lepší přesnost).

Pomocí následujícího vzorce 3.1 nám tato metoda ohodnotí jak hodně se daná okénka shodovaly. Čím menší hodnotu dostaneme, tím byly rozdíly jasů pixelů menší (shoda korespondujících pixelů je pravděpodobnější).

$$\frac{1}{n^2}\sum_{i=-(n-1)/2}^{i=(n-1)/2}\sum_{j=-(n-1)/2}^{j=(n-1)/2}\left(I_L\left(x_L+i,y_L+j\right)-I_R\left(x_R+i,y_R+j\right)\right)^2$$

Rovnice 3.2 Vzorec pro ohodnocení shody SSD metodou

Pojďme si tento vzorec ještě trochu přiblížit:

- Normalizační člen n=velikost zvoleného okénka (n musí být liché číslo)
- x_l,x_r=x souřadnice pixelu z levého a pravého obrazu
- y_l,y_r=y souřadnice pixelu z levého a pravého obrazu
- I_l=hodnota jasu pixelu v levém obraze
- I_r=hodnota jasu pixelu v pravém obraze

Hlavní problém této metody je zvolení správné velikosti zmiňovaného okénka (okolí kolem pixelu). Okénko musí být dostatečně velké na to, aby zahrnulo dostatek variací jasů pro spolehlivé párování, ale zároveň musí být dostatečně malé na to, aby se vyhnulo efektům promítacího narušení. Pokud je zvolené okénko moc malé, ve výsledku dostaneme špatný disparitní odhad, protože se na obrázku vyskytovalo málo variací jasů. Metoda nemohla dostatečně ohodnotit, zda se pixely shodovaly nebo ne. Na druhou stranu, když zvolíme okénko moc velké, také můžeme dostat špatný výsledek, protože toto okénko nám pokryje hodně pixelů, které nám určují hloubku (tzn. pixely, které jdou vidět pouze v levém nebo pravém obraze). Z těchto důvodů se okénko musí zvolit podle jasu a disparit příslušných obrazů. Tímto problémem se zabývá metoda Adaptive window [11].

3.1.2 Sum of Absolute Differences

Druhou nejpoužívanější metodou je SAD. Tato metoda hodnotí shody pixelů pomocí následujícího vzorce [4]. SAD je výpočetně méně náročná než metoda SSD. Proto bychom si ji měli zvolit v aplikacích, kde nám záleží na rychlosti výpočtu. Tato metoda funguje podobně jako SSD, akorát při ohodnocení shody nepoužívá součet mocnin rozdílů jasů pixelů, ale součet absolutních rozdílů jasů. Platí u ní (stejně jako SSD), čím menší hodnotu dostaneme pomocí této metody, tím byly rozdíly jasů pixelů menší (a shoda pixelů byla pravděpodobnější).

$$D_{SAD} = \sum_{(i,j)\in U} \left| I_1(x+i, y+i) - I_2(x+d_x+i, y+d_y+j) \right|$$

Rovnice 3.3 Vzorec pro ohodnocení shody SAD metodou

I když máme hodně metod pro ohodnocení shody pixelů, které můžeme použít, v praxi není jednoduché vybrat tu nejlepší pro danou aplikaci. Pro výběr nejlepší metody, je třeba vyzkoušet více metod a vyhodnotit, která je pro nás ta správná.

3.2 Globální metody

Ačkoli lokální metody jsou lehčí na implementaci, hlavní výhoda globálních metod je kvalitnější zobrazení výsledné disparitní mapy (např. méně chyb). Oproti lokálním metodám, tyto metody berou v úvahu překryté objekty. Z praxe se ukázalo, že globální metody si vedou lépe v korespondenci pixelů v obrazech, kde máme nedostatek textur [4]. Existuje jisté spojení mezi globálními a lokálními metodami. Některé globální metody používají pro výpočet disparitní mapy metody lokální. Nejpoužívanější metody pro globální zpracování jsou:

- Propagace zpráv (*Belief propagation*)
- Grafové řezy (*Graph cut*)
- Maximální tok sítí (*Max-flow*)

Propagace zpráv
Princip této metody spočívá v iterativním zasílání zpráv mezi sousedními uzly (pixely obrazu) tak, aby nám co nejvíce minimalizovala cenovou funkci. Zpráva je zde myšlena jako pravděpodobnost, podle které uzel na základě již dostupných informací od jiných uzlů určí nejvhodnější disparitu. V současné době algoritmy, které využívají tuto metodu podávají nejkvalitnější výsledky.

Grafové řezy
Tato metoda minimalizuje globální cenovou funkcí tím, že vypočítá maximální tok v grafech. Nejprve však musí provést označení pixelů. Jakým způsobem se značení provede máme vysvětlené v [4].

My se zaměříme hlavně na speciální optimalizaci globálních metod a tou je **Dynamické programování.**

4 Dynamické programování

4.1 Úvod

Výraz dynamické programování byl poprvé použit v roce 1940 Richardem Bellmanem. Slovo dynamické bylo použito Bellmanem, protože znělo úchvatně, ne proto, že vysvětluje jak metoda fungovala [14]. Algoritmus řešení je založen na Bellmanově principu optimality: „*Podstrategie optimální strategie je opět optimální*“.

Dynamické programování funguje obdobně jako greedy algoritmus, ale negeneruje se pouze jedná posloupnost. Zkoumají se všechny posloupnosti, které by mohly být optimální a vylučují se ty, které optimální nebudou [17]. Dynamické programování je tedy odvětví optimalizace. Používá se nejenom v programování, ale i v matematice. Pro tuto optimalizaci je charakteristické použití nějaké datové struktury (nejčastěji pole), kam si ukládáme řešení již vyřešených problémů. Metoda je obzvláště vhodná na úlohy, které se dají dělit na podúlohy. Tyto podúlohy jsou si podobné a mohou se opakovat. V mnoha úlohách jde volit způsob rozkladu na podproblémy. Tato volba může mít vliv na efektivitu celého výpočtu. Existují dva způsoby použití dynamického programování:

- Memoization
- Postup zdola nahoru

4.1.1 Memoization

Jedná se o postup shora dolů (tzn. od větších podproblému k menším). Pokud řešení jakéhokoli problému lze formulovat rekurzivně pomocí řešení jeho podproblémů a pokud se jeho podproblémy opakují, potom si můžeme řešení těchto podproblémů někam ukládat (např. do tabulky). Potom kdykoli se budeme pokoušet řešit nový podproblém, nejprve se podíváme do tabulky, zda už jsme tento podproblém nevyřešili. Pokud jsme výsledek našli v tabulce, jednoduše použijeme výsledek z tabulky. Pokud řešení v tabulce nebylo nalezeno, vyřešíme podproblém a výsledek uložíme do tabulky (nebo do nějaké datové struktury). Dále bychom si měli říci, že tento způsob řeší pouze podproblémy, u kterých je potřeba znát řešení. Proto musíme nějak evidovat podproblémy, které už jsme vyřešili.

4.1.2 Postup zdola nahoru

Jedná se o postup od menších podproblémů k větším [12]. Oproti postupu memoization nekontrolujeme, zda jsme problém už řešili nebo ne, ale rovnou saháme pro výsledek, který máme uložený v pomocné tabulce (nebo v jiné datové struktuře). Podproblémy však musíme řešit ve vhodném pořadí. Musí být zaručeno, aby byly vyřešeny menší podproblémy, jejichž řešení potřebujeme k vyřešení větších podproblémů. Kdybychom nepostupovali takhle postupně, došlo by k pádu programu, jelikož řešený podproblém by nemohl být vyřešen kvůli chybějícímu výsledku jeho podproblému. Tento způsob implementace oproti memoization řeší všechny podproblémy, tedy i ty, které nejsou pro výpočet celkového výsledku nutné. Někdy umožňuje efektivnější implementaci než memoization.

Příklad algoritmů, které používají dynamické programování:

- Viterbi algoritmus
- Grafové úlohy a jejich příslušné grafové algoritmy
- Earley algoritmus
- Algoritmy pro řešení problému stereo korespondence

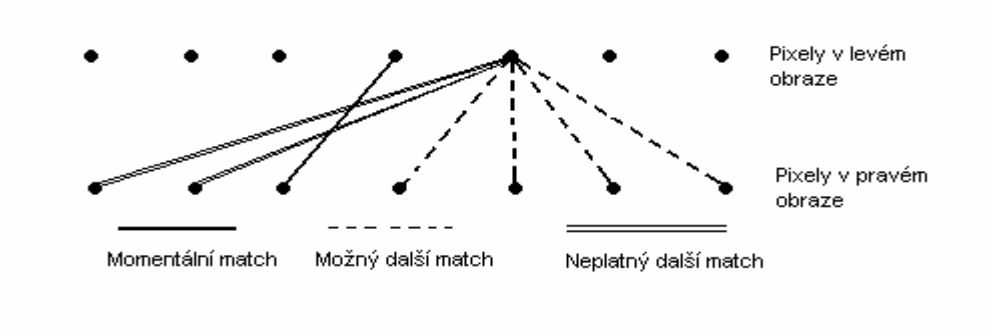
4.2 Stereo korespondence

Předtím, než se podíváme na princip metody dynamického programování, si musíme uvést několik omezení, se kterými tato metoda počítá. Omezení byla objevena v průběhu vývoje metod stereo korespondence. Aby nám metoda tedy vracela správné výsledky, musí mít správně připravená vstupní data.

4.2.1 Omezení

První omezení se nazývá monotonie (*monotonicity*). Spočívá v zachování pořadí pixelů při korespondenci. Vymezuje nám, kde můžeme hledat korespondenci pro daný pixel. Lepší znázornění tohoto omezení můžeme vidět na obrázku 4.1.

.



Obrázek 4.1 Omezení při stereo korespondenci

Pokud nebudeme mít možnost uplatnit omezení monotonie, můžeme uplatnit omezení uspořádání (*ordering constraint*). Pro správný výpočet disparitní mapy vyžaduje optimalizace, založená na dynamickém programování, splnit alespoň jedno z těchto omezení [15]. Vyžadujeme, aby bylo zachováno stejné pořadí pixelů na řádku mezi dvěma obrazy. Tento požadavek nám mohou narušit úzké předměty, nacházející se v popředí obrazu. Pokusím se vysvětlit, jakým způsobem naruší tento předmět v popředí pořadí pixelů. Představme si, že máme obraz, kde v pozadí budeme mít nějaký větší objekt (třeba bednu) a v popředí budeme mít jeden úzký objekt (např. tyč). Narušení spočívá v tom, že na jednom obraze bude celá bedna před touto tyčí a na druhém obraze bude tato tyč rozdělovat bednu na dvě části.

Protože metoda založená na dynamickém programování zpracovává obraz postupně po řádcích, musíme zařídit, aby vstupní obrazy měly rovnoběžné epipoláry. To znamená, že pokud dostaneme na vstupu obrazy, ve kterých řádek pixelů v obraze levém neodpovídá stejnému řádku pixelů v obraze pravém, je potřeba provést nejprve rektifikaci (tuto metodu máme popsanou v kapitole 2).

Další omezení (unikátnost) nám říká, že pixel v obraze levém může korespondovat pouze s jedním pixelem v obraze pravém, nebo se žádným. V případě, že máme určen referenční obraz pravý, platí toto omezení i naopak. Toto omezení nám může selhat v případě, že se v obrazech vyskytují průhledné objekty.

Výsledná disparitní mapa by měla být hladká, takže sousední disparity by neměly moc kolísat. To znamená, že potřebujeme, aby po celém obraze měly sousední pixely co nejméně rozdílné disparity. Toto omezení nazýváme omezení hladkosti (*continuity constraint*).

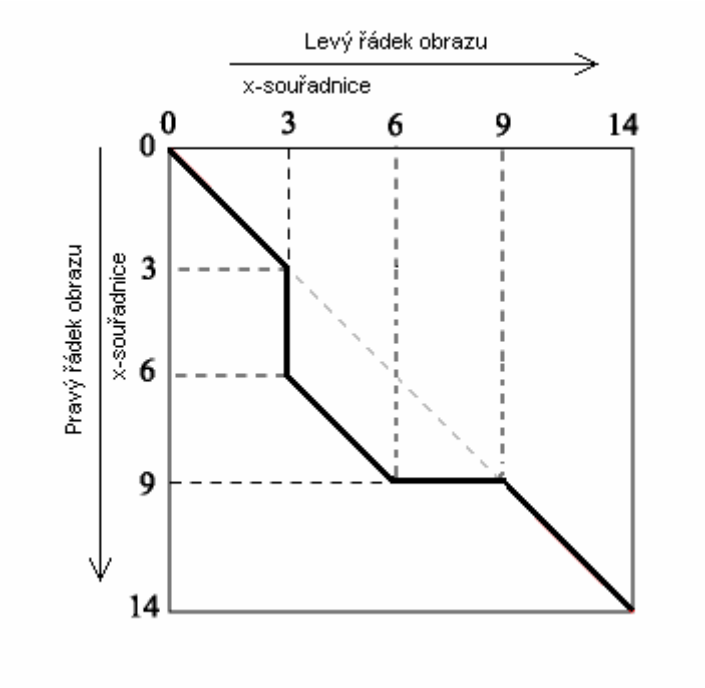
4.2.2 Princip

Díky omezení uspořádání a unikátnosti se nám z procesu párování stává problém nalezení optimální cesty mezi dvěma korespondujícími řádky obrazů. Tato metoda tedy hledá nejlevnější (nejlepší) cestu, která začíná v levém horním rohu grafu a končí v pravém dolním

rohu (viz obrázek 4.2). Na obrázku je nakreslený příklad takové optimální cesty. V každém pixelu máme tři možnosti dalšího kroku:

- Šikmo dolů- Takhle pokračujeme v případě korespondence
- Doprava- Takhle postupujeme v případě, že není nalezena korespondence a pixel lze vidět pouze v levém obraze
- Dolů- Takhle postupujeme v případě, že není nalezena korespondence a pixel lze vidět pouze v pravém obraze

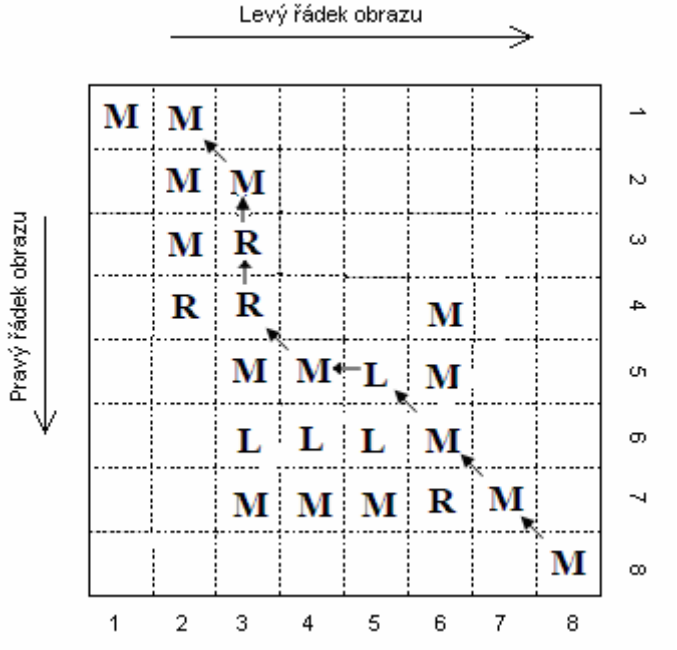
Popíšeme si ještě vliv těchto kroků na disparitu. Pokud jsme nalezli korespondenci, disparita sousedních pixelů se nemění. V případě, že nebyla nalezena korespondence a pixel se nachází v levém obraze, hodnota disparity stoupne. Pokud se pixel nachází v obraze pravém, hodnota disparity daného pixelu klesá.



Obrázek 4.2 Souřadnicová síť

Uvedeme si příklad:
Máme 450 pixelů, v každém pixelu máme tři možnosti, což nám dává $3^{450} = 5*10^{214}$ možných cest. Proto dynamické programování používá matici, která představuje všechny možné kombinace pixelů. Do této matice pro každý pár pixelů se ukládá stav, který měl při kroku z předchozího páru pixelů minimální hodnotu (tzn. ze tří cest, které vedou do tohoto bodu, vybere tu nejlevnější). Tímto krokem eliminujeme nadbytečné cesty, které by se do tohoto bodu také dostaly, ale měly by horší skóre. Optimální cestu si najdeme tak, že začneme v poslední buňce matice a podle stavů buněk postupujeme až na začátek matice (viz obrázek 4.3). V obrázku nejsou vyplněny všechny buňky matice, ale máme znázorněnou pouze výslednou cestu a její okolí. Shodující se pixely máme označeny písmenkem M. Písmenkem R označíme

pixely, které jsou vidět pouze v pravém obraze a písmenkem L označíme hodnoty pixelu, které jsou vidět pouze v obraze levém.



Obrázek 4.3 Ukázka matice s vybranou cestou

Je třeba si ještě něco dodat k postupu při hodnocení stavu. Při korespondenci používá metoda dynamického programování většinou nějakou lokální metodu, která mu tuto korespondenci ohodnotí. Pokud nedojde ke korespondenci, znamená to, že pixel se nachází v levém nebo v pravém obraze (occluded bod) a je potřeba ho ohodnotit tzv. occluded konstantou. Tuto konstantu si programátor volí ručně a to na základě výsledné disparitní mapy. Konstanta musí být zvolena tak, aby nám neovlivňovala správné korespondence. Výsledek dynamického programování je hodně závislý na této konstantě. Při špatné volbě nám vzniká ve výsledku hodně chyb.

4.3 Implementace

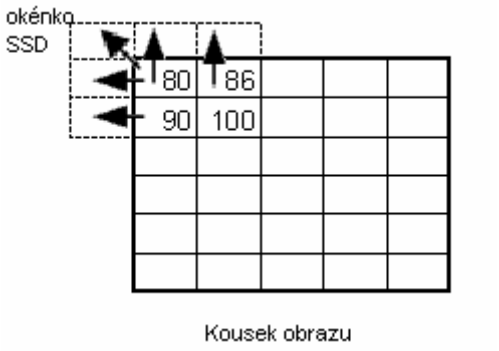
Nyní si ukážeme jak jsme postupovali při implementaci naší metody stereo matche. Jedná se tedy o užití dynamického programování, které využívá pro ohodnocení shody pixelů metodu SSD. Postup si vysvětlíme na menších obrazech, které budou mít šířku obrazu 5 pixelů a výšku 10 pixelů. Velikost okénka metody SSD si zvolíme 3 pixely. Occlusion konstantu si zvolíme 1000. Ještě si musíme vytvořit nový prázdný obraz, o rozměrech vstupních obrazů, který nám bude sloužit jako disparitní mapa. Do něho budeme postupně po řádcích zakreslovat výsledné

velikosti disparit. Dále budeme předpokládat, že vstupní obrazy splňují předpoklady pro použití dynamického programování. Postup si rozdělíme na tyto kroky:

- Příprava obrazů
- Inicializace pole
- Postup vpřed
- Postup vzad

4.3.1 Příprava obrazů

Nejprve si obrazy převedeme na černobílé, protože pak lépe získáme hodnotu intenzity daného pixelu. Tento krok není povinný, protože existují metriky, které umí pracovat s barevnými obrazy. Protože používáme metodu SSD, může nastat jeden problém, který je třeba vyřešit dřív než budeme tuto metodu využívat. Jedná se o problém kdy okénko, které nám kontroluje okolí, dosahuje mimo obraz na daném pixelu (viz obrázek 4.4). Řešíme to tak, že na tato místa dosadíme nejbližší hodnotu pixelu (jas pixelu). V našem případě jsme si vykreslili dva nové obrazy, které budou totožné s původními, ale budou na všech stranách o jeden pixel zvětšeny. Záleží to na velikosti zvoleného okénka, např. při velikosti 5, musí být obraz zvětšený o dva pixely. K těmto obrazům přistupujeme pouze při volání metody SSD, jinak pracujeme stále s původními obrazy.

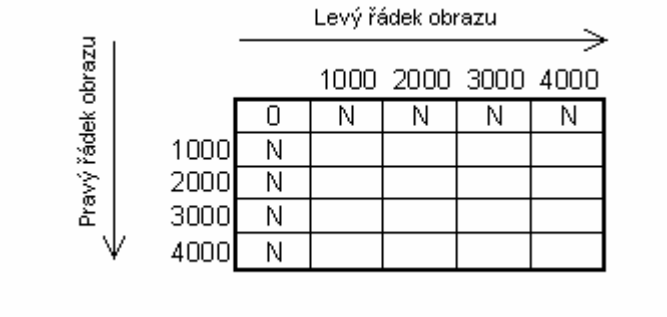


Obrázek 4.4 Problém s okénkem

4.3.2 Inicializace pole

V této chvíli si vytvoříme z velikosti šířky obrazu dvourozměrné pole[r][l]. Index l bude představovat x-souřadnici pixelu v levém obraze a index r bude představovat x-souřadnici pixelu v obraze pravém. Následně si musíme takto připravit pole (viz obrázek 4.5). V buňkách pole si budeme ukládat dvě položky: stav a skóre. Buňky, ve kterých jsme neměnili stav, si označíme písmenkem N (neinicializované). Do první buňky si uložíme hodnotu skóre 0. Poté si vyplníme první řádek a první sloupec. Stavy těchto buněk ponecháme, ale hodnotu skóre

budeme postupně zvedat o naši zvolenou konstantu occlusion. U zbývajících buněk nenastavujeme nic (na obrázku jsou to ty prázdné buňky).

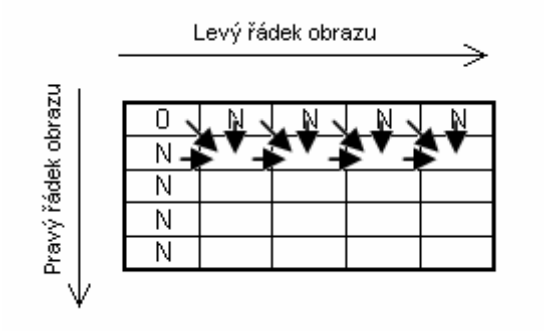


Obrázek 4.5 Inicializace pole

4.3.3 Postup vpřed

Nyní postupně vyplníme u zbývajících buněk jejich stavy a skóre (viz obrázek 4.6). Z obrázku je vidět, že je třeba opravdu postupovat postupně, protože k určení stavu a skóre následné buňky musíme znát skóre předcházejících buněk. Do nové buňky se můžeme dostat ze tří předcházejících buněk. Cestu, pro kterou se rozhodneme, nám určí skóre. V každé buňce provedeme výpočet všech těchto tří cest (tzn. vezmeme skóre předchozí buňky a přičteme k němu hodnotu směru). Ten směr, který bude mít nejmenší skóre pro danou buňku, se do této buňky zapíše jako příslušný stav. Mohou tedy nastat tři stavy buněk a to podle toho, kterým směrem jsme se v poli posunuli (tzn. jakým směrem jsme se dostali k buňce, u které budeme měnit stav a skóre). Označíme si je takto:

- R- pokud jsme se posunuli směrem dolů (pole[r+1][l]), ke skóre přičteme hodnotu occlusin
- L- pokud jsme se posunuli směrem doprava (pole[r][l+1]) a ke skóre přičteme hodnotu occlusion
- M-pokud jsme se posunuli směrem šikmo dolů (pole[r+1][r+1]) a ke skóre přičteme hodnotu výsledku metody SSD



Obrázek 4.6 Vyplnění buňek

4.3.4 Postup vzad

Nyní, když máme vyplněné všechny stavy buněk, můžeme určit výslednou (nejlevnější) cestu. Budeme předpokládat, že vyplněné pole vypadá nějak takto (viz obrázek 4.7). Začneme buňkou, která se nachází v levém dolním rohu (poslední buňka v poli). Zkontrolujeme její stav a podle stavu se přesuneme na další buňku takto:

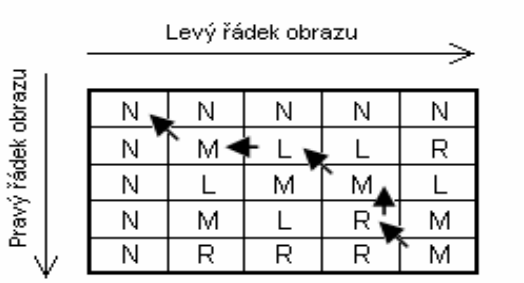
- R- Následná buňka se nachází na pole[r-1][l]
- L- Následná buňka se nachází na pole[r][l-1]
- M-Následná buňka se nachází na pole[r-1][r-1]

Při každém přesunu si uložíme někde do pomocného pole pom[i] index r a index l. To jsou x-souřadnice obrazu, ze kterých pak vypočteme disparitu. Tento krok opakujeme, dokud se nedostaneme k buňce, která má status neinicializovaná. Tímto jsme poskládali výslednou cestu, kterou máme uloženou v pomocném poli. Nyní musíme pro každou dvojici hodnot indexů r a l uloženou v poli pom[i], vypočíst jejich disparitu (viz rovnice 4.1) (v poli postupujeme i=i+2).

$$d = \left| pom[i] - pom[i+1] \right|$$

Rovnice 4.1 Výpočet disparity

Výsledky můžeme ukládat rovnou do naší výsledné disparitní mapy. Avšak musíme si uvědomit, že cesta je poskládaná od posledního pixelu k prvnímu. Proto, až budeme tento řádek ukládat do své disparitní mapy, je třeba hodnoty vkládat od konce pole.



Obrázek 4.7 Výsledná cesta

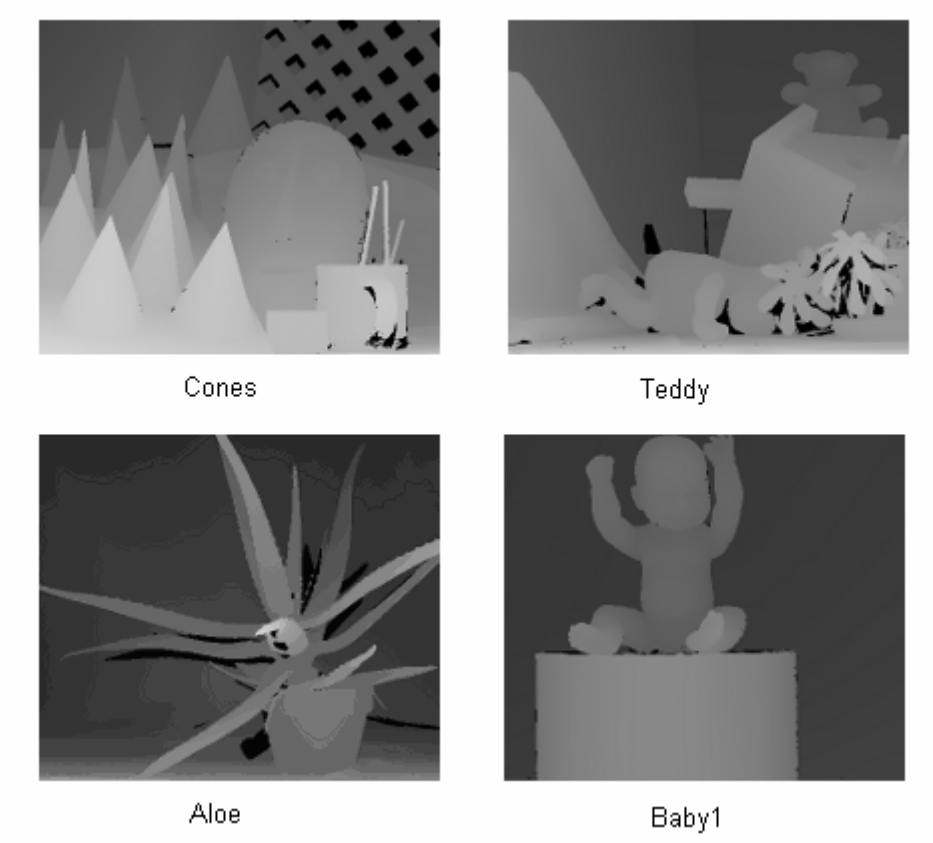
Nyní se nám podařilo nalézt a uložit nejlevnější cestu mezi dvěma obrazy v prvním řádku. Pro vyplnění celé disparitní mapy musíme opakovat následující kroky pro všechny řádky v obraze: **inicializace pole, postup vpřed, postup vzad**.

4.4 Testování

Náš naimplementovaný algoritmus otestujeme na sadě obrázků [1, 2, 3]. Obrázky jsou volně dostupné na stránkách http://vision.middlebury.edu/stereo/data/. Tato stránka dokumentuje metody stereo korespondence (výkon, kvalita).

4.4.1 Vzorové disparitní mapy

Pro každou dvojici testovaných obrázků jsme měli k dispozici výslednou disparitní mapu (viz obrázek 4.8). Podle těchto disparitních map jsme porovnávali naše disparitní mapy.



Obrázek 4.8 Vzorové disparitní mapy

Protože dodané disparitní mapy jsou zobrazené na větší subpixelovou přesnost, před kontrolou jsme si naši disparitní mapu museli upravit. Mapu jsme upravili tak, že jsme vynásobili disparity měřítkem. Toto měřítko jsme převzali od autorů [1, 2, 3]. Pro testování jsme použili scény s názem Teddy, Cones, Aloe, Baby1. Testovali jsme rozdíly výsledné disparitní mapy při různem nastavení occlusion konstanty a velikosti okénka metody SSD. U každé scény si potom vybereme nějakou zajímavost v rozdílu výsledku a tu si pak ukážeme.

4.4.2 Naměřené hodnoty

V první scéně nám vznikl velký rozdíl u testu č.1 a č.2, kdy velikost okénka SSD zůstala stejná, akorát jsme změnili occlusion konstantu (viz tabulka 4.1). Podívejme se, jakým způsobem nám tato konstanta změnila výslednou disparitní mapu (viz obrázek 4.9).

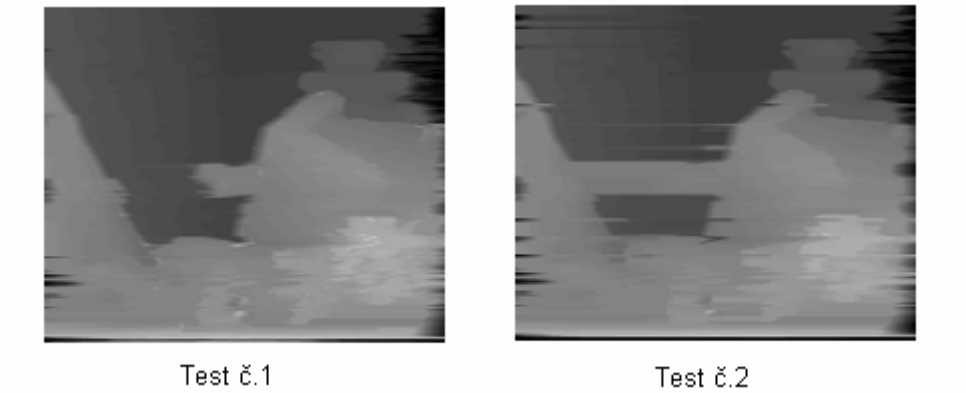
V druhé scéně si ukážeme rozdíl disparitních map (obrázek 4.10), kdy occlusion konstanta zůstane stejná, ale měníme velikost okna metody SSD. Konkrétně se bude jednat o testy č.1 a č.2 (viz tabulka 4.2).

Ve třetí scéně byly rozdíly výsledných disparitních map minimální (viz tabulka 4.3). Rozdíl mezi testem č.1 a testem č.6 můžeme vidět na obrázku 4.11. Velikost většího SSD okna nám způsobila spojení některých listů dohromady.

Pro správné zobrazení poslední scény, bylo potřeba zvolit malou occlusion konstantu. Při větší konstantě se nám obrázek začal úplně rozpadat. Proto máme v tabulce 4.4 rozdíl occlusion konstant minimální. Přesto se to na výsledné disparitní mapě docela dost projevilo (viz obrázek 4.12)

Scéna: Teddy č.testu	Meřítko	Occlusion konstanta	Velikost SSD	Vadné pixely [%]
1	4	400	5	42
2	4	1200	5	56
3	4	400	9	45
4	4	1200	9	47
5	4	400	15	50
6	4	1200	15	53

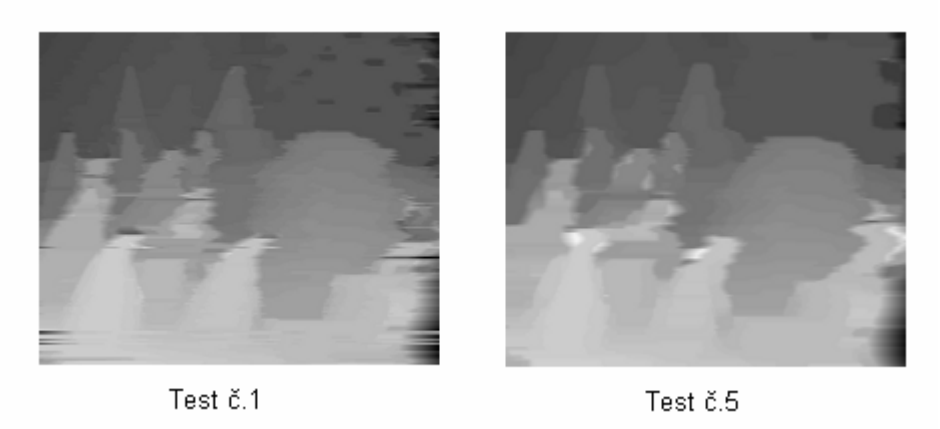
Tabulka 4.1 Naměřené hodnoty, Teddy



Obrázek 4.9 Rozdíl disparitních map, Teddy

Scéna: Cones č.testu	Meřítko	Occlusion konstanta	Velikost SSD	Vadné pixely [%]
1	4	600	5	48
2	4	1200	5	50
3	4	600	9	49
4	4	1200	9	50
5	4	600	15	51
6	4	1200	15	50

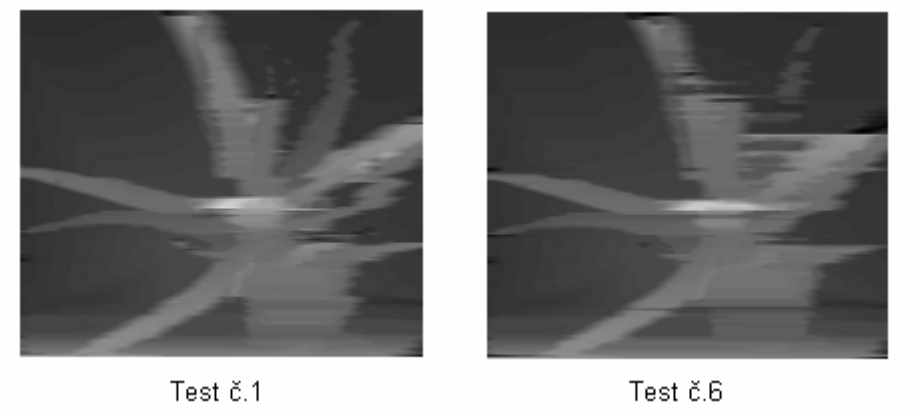
Tabulka 4.2 Naměřené hodnoty, Cones



Obrázek 4.10 Rozdíl disparitních map, Cones

Scéna: Aloe č.testu	Meřítko	Occlusion konstanta	Velikost SSD	Vadné pixely [%]
1	3	400	5	42
2	3	1200	5	43
3	3	400	9	44
4	3	1200	9	44
5	3	400	15	46
6	3	1200	15	46

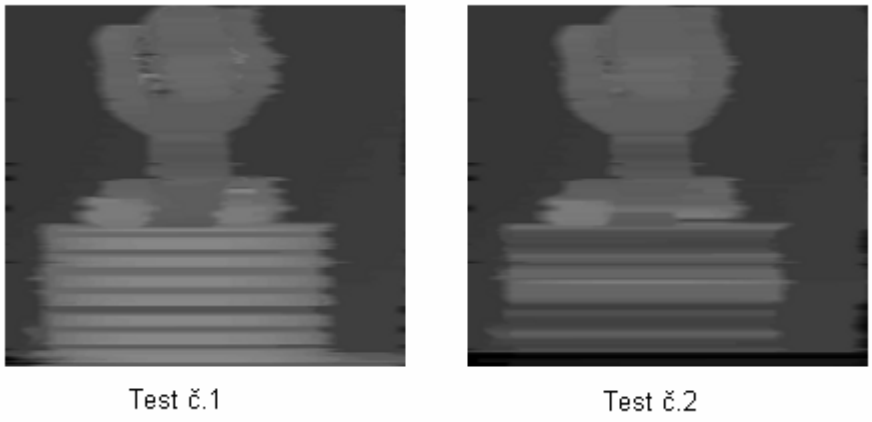
Tabulka 4.3 Naměřené hodnoty, Aloe



Obrázek 4.11 Rozdíl disparitních map, Aloe

Scéna: Baby1 č.testu	Meřítko	Occlusion konstanta	Velikost SSD	Vadné pixely [%]
1	3	200	5	44
2	3	500	5	54
3	3	200	9	47
4	3	500	9	55
5	3	200	15	51
6	3	500	15	54

Tabulka 4.4 Naměřené hodnoty, Baby1



Obrázek 4.12 Rozdíl disparitních map, Baby

Z testování vyplynula jedná typická vlastnost dynamického programování. Řádky v obraze nám občas tzv. „ujedou“. Nejlepších výsledků jsme dosáhli při nastavení okénka SSD na velikost 5. Větší okénka nám způsobila spojení méně vzdálených objektů dohromady. Nastavení occlusion konstanty mělo také velký vliv na výsledek. Při nastavení velkých hodnot jsme způsobili, že pixely korespondovaly i tam, kde už neměly. Objekty na obrázku se nám tak zvětšily.

5 Závěr

Díky této práci jsme se dozvěděli více o problematice stereo korespondence. Vzhledem k rozsáhlosti tohoto problému si myslím, že implementace algoritmu, který bude podávat kvalitní výsledky v reálném čase, bude ještě dlouho trvat. Seznámili jsme se s několika algoritmy, které řeší tento problém. Zaměřili jsme se na jeden z nich, který jsme popsali podrobněji.

Následně jsme naimplementovali algoritmus založený na dynamickém programování. Přesněji se jedná o jednoprůchodové dynamické programování. Pro ohodnocení shody pixelů náš algoritmus používá metodu SSD. Tento algoritmus jsme dále podrobili sérii testů, na kterých jsme prozkoumali výkon a správnost algoritmu. Z testů nám vyplynulo, že správnost výsledku této metody je úzce spojeno s nastavením occlusion konstanty. Konstantu musíme nastavovat ručně, což je u tohoto algoritmu velmi neefektivní. Také jsme zjistili, že algoritmus nepodává kvalitní výsledky.

Pro zlepšení naší metody bychom navrhli aplikovat metodu dynamického programování dvěma průchody. Tím získáme kvalitnější disparitní mapu. To znamená, že při prvním průchodu bychom hledali korespondenci pixelů z levého obrazu v pravém a při druhém z pravého obrazu v levém. Takto lépe zobrazíme zakryté body. Avšak tento způsob vylepšení by nám zvýšil dobu zpracování přibližně na dvojnásobek.

Algoritmus bychom mohli určitě zrychlit několika způsoby. Protože dynamické programování pracuje s řádky obrazu nezávisle na sobě, viděli bychom urychlení programu v implementaci podpory vícejádrových procesorů. Protože jsme měli k dispozici pouze dvoujádrový procesor, upravili bychom algoritmus tak, že jeden procesor by nám zpracovával liché řádky obrazu, zatímco druhý procesor by nám paralelně zpracovával sudé řádky obrazu. Zpracování obrazů by pak mělo být teoreticky o polovinu rychlejší v případě dvoujádrového procesoru. Další způsob urychlení programu vidíme v omezení matice, ze které získáme optimální cestu. Protože se udává u obrázků maximální dovolená disparita, nemusíme počítat v matici se všemi cestami. Proto bychom omezili tuto matici pouze na ty buňky, přes které očekáváme průchod při cestě zpět. Uděláme to přesně tak, že vyplníme pouze ty buňky, které leží v diagonálách vzdálené od středu do vzdálenosti hodnoty maximální disparity na obě strany.

6 Literatura

[1] D. Scharstein a C. Pal.
Learning conditional random fields for stereo. CVPR 2007, Minneapolis, MN, 2007.

[2] D. Scharstein a R. Szeliski.
High-accuracy stereo depth maps using structured light. CVPR2003 , volume 1, pages 195-202, Madison, WI, 2003.

[3] H. Hirschmüller a D. Scharstein.
Evalution of cost functions for stereo matching. CVPR 2007, Minneapolis, MN, 2007.

[4] B. Cyganek a J. Paul Siebert.
An Introduction to 3D Computer Vision Techniques and Algorithms. 1.vyd, 2009, ISBN 978-0-470-01704-3

[5] Holota R., Fiřt J.
Digitalizace a zpracování obrazu. s. 34-38. Plzeň, Západočeská univerzita, 2002, ISBN 80-7082-917-6.

[6] Gábor Blázsovits.
Interaktívna učebnica spracovania obrazu. 1 vyd, Bratislava, 2006, ISBN 80-89186-08-4

[7] P. Fua.
A Parallel Stereo Algorithm that Produces Dense Depth Maps and Preserves Image Feature. 6. vyd, 1993

[8] Štěpán Kuděj
*Zpracování videa na počítači.*České Budějovice, 2005, diplomová práce

[9] Jakub Štilec
Využití počítačového vidění v robotice.Liberec, 2008, disertační práce

[10] Takeo Kanade and M. Okutomi.
A stereo matching algorithm with an adaptive window: theory and experiment. 16. vyd, 1994

[11] http://cs.wikipedia.org/wiki/Zrak [online]

[12] Zdeněk Sawa.
Seminář z programování. 2009, přednáška

[13] http://en.wikipedia.org/wiki/Dynamic_programming [online]

[14] Professor Zhigang Zhu.
Stereo Vision. 2009, přednáška

[15] D. Scharstein and R. Szeliski.
A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. IJCV 2002

[16] http://en.wikipedia.org/wiki/Quantization_%28signal_processing%29 [online]

[17] http://cs.wikipedia.org/wiki/Algoritmus [online]

[18] http://vision.middlebury.edu/stereo/eval/ [online]